Significance-Based Pruning for Reservoir's Neurons in Echo State Networks

Simone Scardapane, Danilo Comminiello, Michele Scarpiniti, and Aurelio Uncini

Department of Information Engineering, Electronics and Telecommunications (DIET),
"Sapienza" University of Rome, via Eudossiana 18, 00184, Rome
{simone.scardapane,danilo.comminiello,
michele.scarpiniti}@uniroma1.it,aurel@ieee.org

Abstract. Echo State Networks (ESNs) are a family of Recurrent Neural Networks (RNNs), that can be trained efficiently and robustly. Their main characteristic is the partitioning of the recurrent part of the network, the *reservoir*, from the non-recurrent part, the latter being the only component which is explicitly trained. To ensure good generalization capabilities, the reservoir is generally built from a large number of neurons, whose connectivity should be designed in a sparse pattern. Recently, we proposed an unsupervised online criterion for performing this sparsification process, based on the idea of *significance* of a synapse, i.e., an approximate measure of its importance in the network. In this paper, we extend our criterion to the direct pruning of neurons inside the reservoir, by defining the significance of a neuron in terms of the significance of its neighboring synapses. Our experimental validation shows that, by combining pruning of neurons and synapses, we are able to obtain an optimally sparse ESN in an efficient way. In addition, we briefly investigate the resulting reservoir's topologies deriving from the application of our procedure.

Keywords: Echo State Networks, Recurrent Neural Networks, Pruning, Least-Square.

1 Introduction

In the machine learning community, *Recurrent Neural Networks* (RNNS) have always attracted a large interest, due to their dynamic behavior [2]. In fact, a RNN implemented in a digital computer can be shown to be as least as powerful as a Turing machine [6]. Hence, in principle it can perform any computation the digital computer can be programmed to. However, the same dynamic behavior has always made RNN training difficult and subject to a large number of theoretical and numerical drawbacks [2].

Over the last two decades, different researchers independently proposed three similar models that later converged in the field of *Reservoir Computing* (RC) [3]. An RC model is a RNN architecture whose processing is partitioned in two components. First, a recurrent network, called *reservoir*, is used to process the input and extract a large number of dynamic features. Then, a static network, called *readout*, is trained on top of these features. In this way, the overall training problem is itself partitioned in two easier subproblems. In particular, in *Echo State Networks* (ESNs), the reservoir is generally

S. Bassis et al. (eds.), *Recent Advances of Neural Networks Models and Applications,*
Smart Innovation, Systems and Technologies 37, DOI: 10.1007/978-3-319-18164-6_4

built with random connections starting from a set of classical analog neurons, while the readout is trained using linear regression techniques [3]. In this way, the original non-linear optimization problem is transformed into a simpler least-square problem, whose solution can be computed efficiently using any linear algebra package.

According to ESN theory, a reservoir has to fulfill three main properties. First, it must be *stable*, in the sense that the effect of any input should vanish after a suitable time. More formally, the reservoir must possess the so-called *echo state property*, which is generally expressed in terms of the spectral radius of its weight matrix [3]. Secondly, the reservoir should be large enough so as to ensure sufficient generalization capabilities. Finally, the connections inside the reservoir (the *synapses*) should be constructed in a sparse fashion, to ensure that the resulting features are suitably heterogeneous. A large amount of research has gone into investigating the echo state property and the optimal sizing of the reservoir [3], while the problem of *sparsification* of the synapses is less explored. Practically, the only criterion in widespread use is to randomly generate only a predefined fraction $d \in [0,1]$ of connections during the initialization of the reservoir. However, the difficulty of choosing an optimal value for d, together with the complete stochasticity of the process, does not lead in general to a significant improvement, which probably explains the large body of works considering fully-connected reservoirs, e.g. [1].

To improve over this, in [5] we introduced an online criterion for generating sparse reservoirs in an unsupervised fashion. The main idea, which is highly inspired to the classical concepts of *Hebbian learning*, is that each synapse has a relative importance in the learning process, which can be approximated well enough by computing an estimate of the linear correlation between its input and output neuron's states. We call this quantity the *significance* of the synapse. Updating the significance at every iteration for all the synapses requires a single outer product between two vectors, hence it does not increase the computational complexity of updating the whole ESN. At fixed intervals, this quantity is used to compute a probability that each synapse is pruned, using a strategy reminiscent of the simulated annealing optimization algorithm [5]. The experimental validation in [5] shows that this procedure is robust to a change of parameters, hence it does not require a complex fine-tuning. Moreover, it provides a significant increase in performance in some situations, which is robust to an increase in the level of memory and non-linearity requested by the task.

One of the questions that remained unanswered in [5] was whether the procedure can be extended directly to the pruning of neurons. This would provide similar advantages with respect to the pruning of synapses, although with one additional benefit, namely, that the reservoir's size itself would adapt during the learning process. Hence, it can potentially free the ESN's designer from choosing an optimal reservoir's size beforehand. In this paper, we answer this question by providing an extension of the concept of significance to the neurons themselves. In particular, we define the significance of a neuron in terms of a weighted average of its neighboring incoming and outgoing connections. Then, a neuron's probability of being deleted is computed in a similar way with what has been said before. We validate our approach by employing the extended polynomial introduced in [1]. Our experiments show that, by combining pruning of neurons

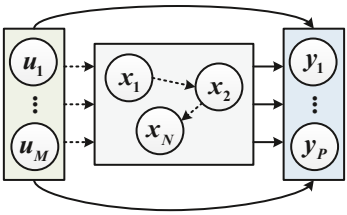


Fig. 1. General schema of an ESN, with no back-connections from the output layer. Fixed and trainable connections are represented with dashed and solid lines respectively.

and synapses, we are able to obtain an optimally sparse reservoir, with a concomitant increase in performance.

The rest of the paper is organized as follows. In Section 2 we briefly introduce the basics of ESN theory. Then, we detail our significance-based pruning for synapses in Section 3. The main novelty of this paper, its extension to the neurons of the reservoir, is in Section 4. Finally, we validate our approach in Section 5. To further investigate our procedure, we analyze the resulting reservoir's topologies in Section 5.3. We conclude with some final remarks in Section 6.

2 Echo State Networks

The ESN used in this work is represented schematically in Fig. 1. It is composed of an input layer of size M, a reservoir of size N, and an output layer of size P. For simplicity, in the following we consider $P = 1$, although everything we say extends naturally to the multi-output case. The connections going from the input layer to the reservoir, and the connections inside the reservoir, are randomly generated at the beginning of the training process. In particular, they are extracted from a normal distribution with unitary variance, in the form of an $N \times M$ matrix $\mathbf{W}_i^r$, and an $N \times N$ matrix $\mathbf{W}_r^r$, respectively. To ensure stability, the latter is then rescaled so as to achieve a predefined spectral radius ρ^* [3]. This is obtained as follows: denoting by ρ the spectral radius of $\mathbf{W}_r^r$, we rescale the original matrix by a factor ρ^*/ρ.

We suppose the network is fed with an input sequence of length S, given by $\{\mathbf{u}(1), \ldots, \mathbf{u}(S)\}$. For example, the i-th input $\mathbf{u}(i)$ may represent a sample of an audio signal, or an element of a spatial sequence. Denoting as $\mathbf{x}(n)$ the N-dimensional vector containing the states of the reservoir's neurons, we update it at every time instant as:

$$\mathbf{x}(n) = f(\mathbf{W}_r^r \mathbf{x}(n-1) + \mathbf{W}_i^r \mathbf{u}(n)) \tag{1}$$

where $f(\cdot)$ is the activation function of the neurons inside the reservoir. In this work, we use $f(\cdot) = \tanh(\cdot)$. The output of the network is computed similarly:

$$y(n) = \mathbf{w}_r^o \mathbf{x}(n) + \mathbf{w}_i^o \mathbf{u}(n) \tag{2}$$

where $\mathbf{w}_r^o$ is the N-dimensional vector linking the reservoir to the output and $\mathbf{w}_i^o$ the M-dimensional vector connecting the input to the output layer. In this work, we consider

the case of linear output, so there is no activation function in Eq. (2). In Fig. 1 we represented fixed connection with solid lines, whilst the trainable connections are shown with dashed lines. In particular, we are interested in learning the vectors $\mathbf{w}_r^o$ and $\mathbf{w}_i^o$. Denote by $\mathbf{d} = [d(1), \ldots, d(S)]$ the concatenation of all the desired outputs, by $\mathbf{s}(n)$ the "extended" state $\mathbf{s}(n) = \left[\mathbf{u}(n)^T \, \mathbf{x}(n)^T\right]^T$ and by $\mathbf{A}$ the concatenation of all such states $\mathbf{A} = [\mathbf{s}(1), \ldots, \mathbf{s}(S)]$. The optimal weights are given by solving the following regularized least-square optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^{P+N \times 1}} \|\mathbf{d} - \mathbf{w}\mathbf{A}\|^2 + \lambda \|\mathbf{w}\|^2 \tag{3}$$

where $\lambda \in \mathbb{R}^+$ is a positive scalar balancing the two terms, and $\|\cdot\|$ denotes the L_2-norm of a vector. Provided we have enough samples, the solution to (3) is given by:

$$\mathbf{w} = \left(\mathbf{A}^T\mathbf{A} + \lambda \mathbf{I}\right)^{-1} \mathbf{A}^T \mathbf{d} \tag{4}$$

where $\mathbf{I}$ is the identity matrix. Practically, the initial values produced by the network are discarded due to their transient state, and are denoted as *dropout* elements. Moreover, multiple sequences in input (e.g., multiple audio signals) are handled by concatenating the resulting matrices.

3 Significance-Based Pruning for the Reservoir's Connections

In this section, we describe our pruning strategy for the reservoir's connections that we introduced in [5]. The strategy acts during the computation of the network states, and it does not require the evaluation of the error gradient. Loosely speaking, it can be seen as an hard thresholded version of the Hebbian rule, and it is inspired to some biologically existing processes in the brain [5]. The main idea is to consider the importance of a synapse (or the *significance*, as we denote it) in terms of the correlation between its input and output neurons. Practically, we define the significance of a synapse at time instant n as:

$$s_{ij}(n) = \frac{1}{T} \sum_{z=n-T}^{n} \frac{(x_i(z-1) - \hat{\mu}_x)(x_j(z) - \hat{\mu}_x)}{\hat{\sigma}_x^2} \tag{5}$$

where T is a time-interval chosen *a priori*, and $\hat{\mu}_x$ and $\hat{\sigma}_x$ are the empirical estimations of the mean and standard deviation of the neuron states. For simplicity, we suppose these are equivalent for all the neurons. The quantities $s_{ij}(n)$ are used to define the probability that a synapse is removed as:

$$p_{ij}(n) = \exp\left\{-\frac{|s_{ij}(n)|}{t(n)}\right\} \tag{6}$$

where $t(\cdot)$ is a positive, monotonically decreasing function of n. This is used to ensure that the probability of removing a synapse is maximal in the beginning of learning and goes to 0 afterwards. This is inspired from the Simulated Annealing optimization algorithm [5], and for this reason we adopt the corresponding terminology and call $t(n)$ the *temperature* of the system. Successively, every Q time instants, we prune each

Algorithm 1. Pseudo-code for a single update step of the pruned ESN, with direct pruning of synapses and neurons.

Data: Input signal $\mathbf{x}(n)$, desired output $d(n)$.
$\mathbf{x}(n) = f(\mathbf{W}_r^r \mathbf{x}(n-1) + \mathbf{W}_i^r \mathbf{u}(n))$
Update sums in Eqs. (5) and (8)
if mod $(n, Q) = 0$ **then**
  $p_{ij}(n) = \exp\left\{-\frac{|s_{ij}(n)|}{t(n)}\right\}$
  $p_i(n) = \exp\left\{-\frac{|s_i(n)|}{\hat{t}(n)}\right\}$
  Prune each synapse with probability $p_{ij}(n)$
  Prune each neuron with probability $p_i(n)$
end

synapse with a probability given exactly by Eq. (6). We use an exponential profile for the temperature $t(n)$:

$$t(n) = \alpha^{(n/Q)-1} t_0 \tag{7}$$

where t_0 is the initial temperature, given *a priori*, and α is called the *scaling factor*. It can be seen from Eq. (7) that the temperature is scaled by a factor α at every "pruning step", given by $(n/Q) - 1$.

4 Extending Pruning to the Reservoir's Neurons

The pruning strategy introduced in the last section can be used to delete unnecessary connections inside the reservoir, hence promoting sparsity. In this section, we show how it can be extended to the direct pruning of neurons. In particular, denote as $\mathscr{I}_j(n)$ the set of *incoming* synapses of the j-th neuron at time-instant n, and by $\mathscr{O}_j(n)$ the set of *outgoing* synapses. The significance of the neuron is defined as:

$$s_j(n) = \frac{1}{2|\mathscr{I}_j(n)|} \sum_{z \in \mathscr{I}_j(n)} s_{jz}(n) + \frac{1}{2|\mathscr{O}_j(n)|} \sum_{z \in \mathscr{O}_j(n)} s_{zj}(n) \tag{8}$$

where $|\cdot|$ denotes the cardinality of the set. Thus, the significance of the neuron is defined as a weighted average of the significance of its neighboring synapses. In this way, neurons belonging to less "significant" clusters, i.e. whose connections are not significant in the sense of Eq. (5), will be denoted by a small value of Eq. (8). We can use this quantity to prune neurons in a similar way with respect to the last section. In particular, every Q time instants we define the probability of removing a given node as:

$$p_i(n) = \exp\left\{-\frac{|s_i(n)|}{\hat{t}(n)}\right\} \tag{9}$$

The new temperature $\hat{t}(n)$ must respect the same properties depicted in the last section. Practically, in all our experiments we use the exponential profile defined by Eq. (7). The overall algorithm, inclusive of pruning of the neurons and of the synapses, is summarized in Algorithm 1.

5 Experimental Validation

5.1 Experimental Setup

To test the efficacy of our strategy, we consider the extended polynomial detailed in [1], which we already adopted previously in [5]. The input to the system consists in one random number extracted from a uniform distribution over $[-1,+1]$. The output is given by:

$$y(n) = \sum_{i=0}^{p} \sum_{j=0}^{p-i} c_{ij} u^i(n) u^j(n-d) \tag{10}$$

where the coefficients c_{ij} are randomly distributed over the same distribution as the input data, and the two parameters p and d control the requirements of the task in term of memory and non-linearity. In particular, increasing p increases the power of the polynomial, while increasing d extends its delay. We consider a reservoir with $N = 250$ neurons in the reservoir. The input-to-reservoir matrix is initialized with full connectivity and weights extracted uniformly from the set $\{-0.1, 0.1\}$. The reservoir weights are extracted from a normal Gaussian distribution, and then $\mathbf{W}_r^r$ is rescaled to have a spectral radius of 0.9, following the results of [1]. We generate 20 sequences of 1000 elements each according to Eq. (10), with the addition of Gaussian noise with variance 0.01. To compute performance, we perform a 10-fold cross validation over the sequences, i.e., at every fold we use 18 sequences for training and the 2 remaining sequences for testing. We prune the network every 100 time instants, and we set $T = Q = 100$. The optimal regularization factor λ in Eq. (4) is found to be around 0.001.

Regarding our strategy, we compare the performance (i) without pruning, (ii) with pruning of the synapses, (iii) with pruning of the neurons, and (iv) with pruning of both simultaneously. In all our experiments, after some trials we set the initial temperature to 0.3. By performing an inner 3-fold cross validation for the scaling factor, we found that the optimal values are around 0.9 for the pruning of the synapses, and 0.5 for the pruning of the neurons.

5.2 Generalization Performance

We perform experiments when increasing simultaneously the power and the delay of the polynomial from 1 to 9. The *Mean-Squared Error* (MSE) averaged over the 10 folds is shown in Fig. 2. We see that the sparse versions of the reservoir have a significant decrease in testing error, which becomes more pronounced for high levels of memory and non-linearity. Moreover, the same testing error is achieved when pruning only synapses, only neurons, or both. As an example, for $p = d = 9$, we have that the sparse ESNs obtain an MSE of 0.53, compared to the original MSE of 0.87, with a 30% decrease approximately.

The original ESN is built from 250 neurons in the reservoir, and full connectivity, for a total of 62500 connections. The ESN with synapse pruning has instead an average number of 7700 connections, i.e., slightly more than 12% of the original one. Similarly, the ESN with pruning of the neuron has a final number of neurons around

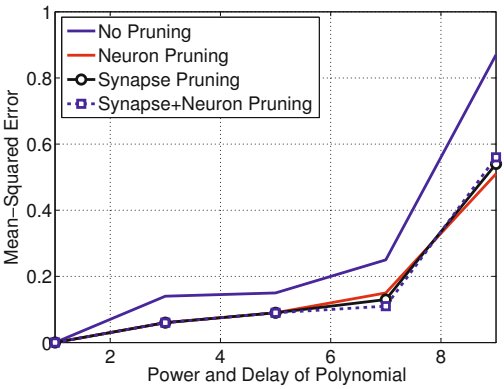


Fig. 2. Average MSE in the four cases under consideration, when increasing simultaneously the delay and power of the polynomial from 1 to 9

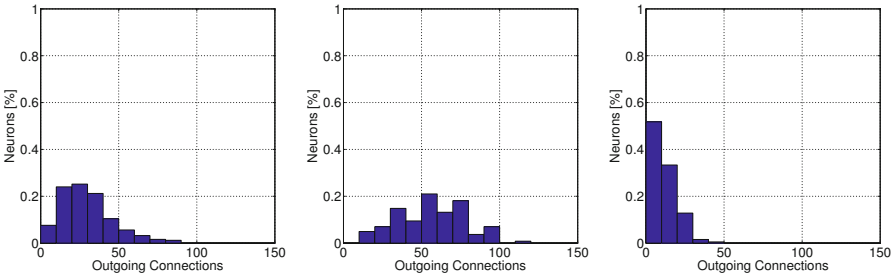
Table 1. Resulting number of neurons and synapses when using the different pruning strategies

Case	Neurons	Synapses
Original	250	62500
Synapse pruning	250	7700
Neuron pruning	110	12000
Full pruning	50	1700

110, with approximately 12000 connections. This means that, together with an increase in performance, the ESN is also faster to train, and easier to eventually implement on an hardware platform. The largest advantage, however, is obtained when both pruning strategies are applied simultaneously. In this case, the final ESN has an approximate number of 50 neurons, with 1700 connections left. This is summarized in Table 1.

5.3 Analysis of the Reservoirs Topology

Before concluding, we briefly investigate an interesting aspect of our strategies, namely, the resulting topologies of the reservoir after pruning. In Fig. 3 we plotted the histograms of the average number of outgoing connections, the so-called *outdegree* [4], inside the reservoirs, after applying the three criteria. We see from Fig. 3-(a) that, when pruning only the synapses, most of the resulting neurons have a relatively small outdegree (ranging in 10-30), and the outdegree decays linearly. The distribution when pruning the neurons is slightly more complex, and is depicted in Fig. 3-(b). We can see that the outdegree has a set of three peaks which are evenly distributed, then decays linearly in both verses. The most interesting aspect, however, is relative to the distribution when pruning both neurons and synapses simultaneously, depicted in Fig. 3-(c). We see that most neurons have a very small number of outgoing connections ($10 - 20$), and a



(a) Synapse pruning (b) Neuron pruning (c) Simultaneous pruning of neurons and synapses

Fig. 3. Histogram of the average number of outgoing connections in the ESN, after applying the three pruning strategies

smaller fraction an outdegree of 30. Hence, it seems that the overall strategy is indeed creating compact "clusters" of neurons. This is an interesting behavior that we are eager to investigate more deeply in a future work.

6 Conclusions

Echo State Networks allows for an efficient training of Recurrent Neural Networks in real-world applications. Due to their nature, however, they generally require large networks, which may be non-applicable in realistic contexts. In this paper, we have extended an algorithm that we proposed for the pruning of the synapses, to the direct pruning of neurons. Our results show that, when the two strategies are applied simultaneously, we are able to obtain optimally sparse reservoir without increasing the computational complexity of the training process.

References

1. Butcher, J., Verstraeten, D., Schrauwen, B., Day, C., Haycock, P.: Reservoir computing and extreme learning machines for non-linear time-series data analysis. Neural Networks 38, 76–89 (2013)
2. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks - with an erratum note. Tech. rep. (2001)
3. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Science Review 3(3), 127–149 (2009)
4. Newman, M.: Networks: an introduction. Oxford University Press (2010)
5. Scardapane, S., Nocco, G., Comminiello, D., Scarpiniti, M., Uncini, A.: An effective criterion for pruning reservoir's connections in echo state networks. In: 2014 International Joint Conference in Neural Networks, pp. 1205–1212 (2014)
6. Siegelmann, H.T.: Neural and super-turing computing. Minds and Machines 13(1), 103–114 (2003)